APPLICATION FOR UNITED STATES LETTERS PATENT

For

NOVEL DYNAMIC OBJECT LIBRARY SOFTWARE ARCHITECTURE

Inventor(s):

Vadim Antonov

Mikhail Kourjanski

Prepared by:

BLAKELY SOKOLOFF TAYLOR & ZAFMAN LLP
12400 Wilshire Boulevard
Seventh Floor
Los Angeles, California 90025-1026

Attorney's Docket No.: 005642.P001

Novel Dynamic Object Library Software Architecture

The present application claims priority to the provisional filed application entitled *Novel Dynamic Object Library Software Architecture*, filed on May 21, 2001, serial no. 60/292,834, which is also incorporated herein by reference.

FIELD OF THE INVENTION

[0001] The present invention relates to the field of software architecture. In particular, the present invention discloses a method for dynamically matching a server and a client over a secure connection and providing information asynchronously between the client and the server, while utilizing a flow control of the information.

BACKGROUND

[0002] Large data centers that are used to transmit customer information, credit card information or the like inherently have many problems. For most of the data centers, the biggest issues are lack of security, decreased quality of performance, scalability issues, and version flexibility to support application sharing among customers (multi-tenancy).

[0003] As the number of customers of a data center increases or if a data center has multiple locations serving multiple customers, security may decrease. Security issues are of great importance for these data centers that serve multiple clients (i.e., banking and financial institutions), as they may have competitors who are generally "hostile" toward one another. It is therefore necessary to provide a secure connection that allows client information to be sent through these data centers without compromising the security of the information.

[0004] With data centers being spread over multiple locations, dramatic decreases in

guaranteed performance may also occur. It is necessary for a system in which shared use of a data center in multiple locations does not diminish performance.

[0005] A further issue with respect to the larger data centers is the problem of adaptability. In an application-sharing model, small customers share the same application. As customer use grows, the data centers must then be able to adapt to this increased growth. Unless adaptation can take place, the economies of scale will not allow for decreased costs. However, if the data centers are able to expand easily, the sharing of the applications becomes much more cost effective.

[0006] Version flexibility may also become a large problem because various customers may have different software version requirements, thus requiring multiple versions to be in use simultaneously. The architecture must be able to adapt to a variety of customer versions or the sharing of applications cannot work well. Additionally, the upgrade policy within the data centers must be flexible enough to allow client software to remain intact while server software is upgraded because it is unreasonable to demand a synchronous upgrade of all client software in the ASP setting.

[0007] Various software applications, including CORBA, Microsoft COM/DCOM, Enterprise Java Beans, and others, have attempted to address these problems. But none of these applications have been able to satisfactorily address all problems.

[0008] Thus a need exists for a new software architecture that avoids those shortcomings, and allows information to be transmitted securely between a client and a server.

SUMMARY OF THE INVENTION

[0009] In accordance with the teachings of the present invention, a method for dynamically matching a first and a second process, the first and second processes being matched using a library, the first and second processes utilizing a secure flow control of information and being connected asynchronously, is disclosed.

[0010] Other objects, features, and advantages of the present invention will be apparent from the accompanying drawings and from the detailed description that follows below.

BRIEF DESCRIPTION OF THE DRAWINGS

[0011] The present invention will be described by way of exemplary embodiments, but not limitations, illustrated in the accompanying drawings in which like references denote similar elements, and in which:

[0012] **Figure 1** is an overview illustration of a structure in which the Exigen Object Library (EOL) would function between a transport layer and an application, according to one embodiment.

[0013] **Figure 2** is an illustration of a push and pull of information between a client and a server, according to one embodiment.

[0014] **Figure 3** is an illustration of a compiler that compiles a new object into a finished EOL object, according to one embodiment.

[0015] **Figure 4** is an illustration of a security model that allows for a secure connection between a client (first process) and a server (second process), according to one embodiment.

[0016] **Figure 5** is an illustration of an implementation of compatibility between a server and a client, according to one embodiment.

DETAILED DESCRIPTION OF THE INVENTION

[0017] In the following description, various aspects of the present invention will be described. However, it will be apparent to those skilled in the art that the present invention may be practiced with only some or all aspects of the present invention. For purposes of explanation, specific configurations are set forth in order to provide a thorough understanding of the present invention. However, it will also be apparent to one skilled in the art that the present invention may be practiced without the specific details. In other instances, well-known features are omitted or simplified in order not to obscure the present invention.

[0018] Referring now to **Figure 1**, wherein an overview illustration of a structure in which the Exigen Object Library (EOL) would function between a transport layer and an application in accordance with one embodiment is shown. As illustrated, the structure 100 contains a running application 101, which uses an EOL application program interface (API), defined in detail in Appendix E. Beneath the running application is business computing services management 102. Below the business computing services management 102 are input channel 103 and output channel 104, functional compatibility modules (objects) 105 and services such as logic, authentication, security issues, etc. 106a-n. Beneath the input and output channels 103 and 104, the functional compatibility modules 105 and the services 106a-n is the Exigen Object Library (EOL) 107 that stores information for creating objects. The application program 101 uses the EOL 107 to access services 106a-n such as server computer authentication and the like. The EOL 107 is immediately above a transport layer 108 that is the bottom of the structure 100. The topology of the transport layer 108 may depend on the actual network, but in most

cases it would typically be an IP network-type topology. Thus layers 107 and 106a-n provide an underlying foundation for the computing services 102 and applications 101 running on top.

[0019] The structure 100 is a message passing software architecture that allows information to be passed between a client (a first process) and a server (a second process). The transport layer 108 provides the connection between the client and the server so information may be passed.

[0020] Appendix A contains a complete description of the Exigen Object Library (EOL); and Appendix B describes the Exigen Object Definition Language (EODL), a novel definition language that defines elements of the EOL. An advantage of the EOL is that objects created by the EOL are always stateful. This is greatly beneficial because there is no known way to consistently implement a security mechanism with stateless objects. These shall be deemed examples of an implementation for illustration, and not as the only way to implement the invention.

[0021] As is typical in a computing environment, **Figure 2** shows a client-server scenario, where a client 201 can either push 211 or pull 210 information from a server 200. A process using the EOL 107 may create an object and become a server 200 for that object. The server 200 creates references to that object, passes those references to other processes and the processes then become clients 201 for that server object. The advantage of the EOL architecture is that it allows for dynamic type matching between the server 200 and the client 201, allowing for a compatibility match. Certain rules, described in appendices A and B, govern this dynamic type matching.

[0022] As an example, most object interface changes add new methods or fields. The object types that differ structurally may still be compatible, allowing the client 201 to safely continue to interact with the newer server 200. However, an object may change behavior in ways that break compatibility between the client 201 and the server 200. To ensure compatibility, type structures are compared. When client 201 and server 200 have object types with the same name and same behavior version numbers, they are considered to be of the same type and compatibility is acknowledged. Due to dynamic matching, no mandatory upgrade of the client side 201 is required or necessary, so therefore it is easier for one software instance to operate on different data sets, and vice versa. Also, scripts can be handed over dynamically, which allows execution of a script on a client 201 pushed 211 by a server 200. If the object types are matched from the server 200 to the client 201 and found compatible, then information may be pushed 211 or pulled 210 between them.

[0023] Another very important and distinctive aspect of the novel art of this disclosure is that instead of using a fixed connection, a flow control is relied upon from the transport layer 108. When information is pushed 211 or pulled 210 from the client 201 to the server 200, the flow control provides a buffer at the origin of the flow so that there is no overflow of information to the recipient of the flow. For example, if a client 201 is sending information to a server 200, the flow control makes sure that the information does not overflow the server 200. The flow control chokes the stream going into the server 200, and the information provided by the client 201 is backed up on the client side 201, rather than the server side 200. This novel functionality allows data centers to be spread out over multiple locations, providing services to multiple users,

while not overloading a server 200 with information from one specific client 201. Hence, such novel functionality adds an additional safety valve against clogging of multisite processing.

[0024] **Figure 3** shows a one-pass compiler 300 that can take a new object 305 and compile it in a single pass into a finished EOL object 310. The finished object 310 can then be embedded in the EOL layer 107 of Figure 1.

[0025] As described in detail in Appendix B, sections 3 – 10.5, the EODL has descriptions for the following items: basic types, scripts, data streams (meaning open channels), dynamic state structure, security, cookies, and object reference.

[0026] **Figure 4** describes the security model of the present invention. Security from clogging of information again relies upon the flow control previously described. Both the client 201 and the server 200 may check their respective security rights with an EOL name server 400. In another embodiment, the client 201 and server 200 may check their respective security rights with separate name servers. When a client 201 wants to access a server-side object, the client 201 needs a reference to that object. The service authentication process (one of the services 106a-n in Figure 1) produces an initial "root" name server object. The name server system 400 locks up the requested object and gives the client 201 a cookie in return for verification. When the client 201 gets permission from the name server 400, the server 200 from which the client 201 wants to request services can verify permission directly from that name server 400. Once the security rights have been checked for the client 201 and the server 200, the client 201 and server 200 may communicate with each other by either pushing 211 or pulling 210 information.

[0027] The lifecycle of the server object is decided by the server 200 itself – the client 201 does not manage the lifecycle.

[0028] If access permission is authorized, the object references may be stored or passed between different client 201 processes. The security model also includes a version compatibility check, so that a client 201 can only reach those versions for which it has been cleared.

[0029] Appendix C further describes the cooperation between server 200 and client 201 objects; and Appendix D describes the nameserver system, which is rather novel.

[0030] **Figure 5** shows an implementation of enhanced compatibility using a tree of nameservers and directories. Nameserver-1 510 holds the root (namespace) 500 of the system, which may have multiple layers of directories. In this example, directory 511, within nameserver-1 510, has its own root.

[0031] Subdirectories 512a and 512b manage objects 513a-n. One of the directories 512b in nameserver-1 510 then forms the root for nameserver-2 520. Nameserver-2 520 has its own internal root 521, and an additional layer of subdirectories 522a-n that manage objects 523a-n. Therefore, in essence, additional nameservers, directories, functional compatibility management servers, etc., are just additional clients to the root directory server. Each object 513a-n, 523a-n has a given name. The directory and nameserver managing that object then know the rights and locking capabilities of that object.

[0032] An application (i.e., activation manager application) is first run which uses the EOL to access a service such as computer authentication. A reference is then returned to the server's operational namespace 500 that contains the activation manager's

configuration for the server and the references for the namespaces of individual server processes. The server process obtains its configuration from its namespace 500, as delegated by the activation manager, and performs initialization. Then using references to the user root nameserver from the configuration objects, the server process registers its service objects with these root nameservers, thus exposing the resources to users.

[0033] A further inventive aspect of this disclosure is that EOL programming provides a consistent way for asynchronous single-thread server implementations. The method invocation of an object, using EOL, is asynchronous (nonblocking) – while the server waits for the results, it can go on with other events in order to be as efficient as possible.

[0034] The above embodiments can also be stored on a device or machine-readable and be read by a machine to perform instructions. The machine-readable medium includes any mechanism that provides (i.e., stores and/or transmits) information in a form readable by a machine (e.g., a computer). For example, a machine-readable medium includes read only memory (ROM); random access memory (RAM); magnetic disk storage media; optical storage media; flash memory devices; electrical, optical, acoustical or other form of propagated signals (e.g., carrier waves, infrared signals, digital signals, etc.). The device or machine-readable medium may include a solid state memory device and/or a rotating magnetic or optical disk. The device or machine-readable medium may be distributed when partitions of instructions have been separated into different machines, such as across an interconnection of computers.

[0035] While certain exemplary embodiments have been described and shown in the accompanying drawings, it is to be understood that such embodiments are merely

illustrative of and not restrictive on the broad invention, and that this invention not be limited to the specific constructions and arrangements shown and described, since various other modifications may occur to those ordinarily skilled in the art.